Engineer

control de calidad obligatorio

Comprobaciones que el código debe pasar antes de avanzar

Qué es

Un control de calidad obligatorio es un conjunto de condiciones que un cambio debe cumplir antes de pasar a la etapa siguiente, con un único resultado al final: aprobado o rechazado. La etapa puede ser un commit, una fusión con la rama principal, una publicación o un despliegue. Las condiciones son cualquier cosa que una máquina pueda decidir: una ejecución del linter sin incidencias, pruebas que pasan, una cifra de cobertura o una compilación que termina.

La palabra decisiva es obligatorio. Una comprobación informa; un control obligatorio decide, y aquello que controla el paso siguiente hace cumplir la decisión. Un linter que imprime cuarenta advertencias en un registro es una comprobación. Ese mismo linter, conectado de forma que el botón de fusión permanezca deshabilitado hasta que su salida no tenga incidencias, forma parte de un control obligatorio.

SonarQube Cloud, donde el término inglés quality gate da nombre a una función, describe este control como un conjunto de condiciones con las que se mide el código durante el análisis. Cada condición combina una métrica, un operador de comparación y un valor de error: «problemas de fiabilidad superiores a 0» es una condición, y si se cumple cualquiera de ellas, el control falla. Cada condición se aplica al código nuevo o al código en su conjunto.

Las plataformas de alojamiento incorporan la misma idea a su proceso de fusión:

- **GitHub** tiene el ajuste de protección de ramas «Require status checks before merging», que exige comprobaciones de estado antes de fusionar. Una comprobación obligatoria debe comunicar un estado satisfactorio, omitido o neutro antes de que los cambios lleguen a la rama protegida.

- **GitLab** tiene una comprobación de fusión llamada «Pipelines must succeed», que exige que los flujos de procesamiento terminen correctamente. Una vez seleccionada, una solicitud de fusión no puede fusionarse mientras su flujo esté fallando, ni tampoco cuando no exista ningún flujo.
- **En el equipo del propio desarrollador**, pre-commit instala comprobaciones como hooks de Git que se ejecutan al confirmar cambios o al enviarlos, y un ejecutor de tareas como Task reúne todo el conjunto bajo un único nombre.

Utilice el concepto en una decisión

Aplique «¿Qué es un control de calidad obligatorio?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cambios probados, publicados y reversibles, para que lleguen a sus clientes un martes por la tarde en vez de un viernes por la noche.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Despliegue GIS SaaS con GitHub Actions — despliegue, datos e informes automatizados

Despliegue GIS SaaS con GitHub Actions Póngase en contacto →

Por qué importa

Una norma que solo existe en un documento se negocia excepción por excepción. Una norma que hace fallar la compilación la sigue todo el mundo, incluida la persona que la escribió, la tarde en que preferiría saltársela. Un control obligatorio convierte las reglas acordadas en algo ejecutable, de modo que cumplirlas deja de depender de la memoria y del estado de ánimo.

También cambia la finalidad de la revisión humana. Cuando una máquina ya ha detectado problemas de formato, importaciones sin usar, enlaces rotos y secretos filtrados, quien revisa puede dedicar su atención a la intención y al diseño, algo que una máquina no puede juzgar. El nivel exigido también es el mismo para cada autor.

La idea de evaluar el código nuevo permite adoptar un control obligatorio en una base de código antigua. Sonar way, el control predeterminado de SonarQube Cloud, se centra en el código nuevo en lugar de corregir el antiguo, con condiciones como una cobertura de pruebas de al menos el 80,0 % y una duplicación de como máximo el 3,0 % en el código nuevo. En una solicitud de cambios solo se utilizan las condiciones relativas al código nuevo. Un proyecto con miles de problemas existentes puede activar el control y exigir ese nivel a cada cambio nuevo sin tener que resolver antes todo lo pendiente.

La velocidad determina si la gente mantiene activado el control. Task ejecuta las dependencias de una tarea en paralelo, de modo que un control formado por muchas comprobaciones independientes puede ejecutarlas simultáneamente en lugar de una tras otra. También ayuda dividir las comprobaciones por coste: la mitad de menor coste en cada commit y el conjunto completo antes de que el código salga del equipo.

Este sitio tiene su propio control obligatorio: `task check`, ejecutado en el equipo donde se escribe el sitio en lugar de en un servicio de compilación alojado. Cada diagnóstico cuenta como error, el generador del sitio se ejecuta con `--panicOnWarning` para que incluso un aviso de obsolescencia detenga la compilación, y se prohíben las supresiones en línea como `# noqa` o `stylelint-disable`, de modo que cada fallo se corrige en su origen en lugar de silenciarlo. Un hook de pre-commit ejecuta las comprobaciones por archivo en cada commit, y un hook de pre-push ejecuta el `task check` completo, que tarda minutos en vez de segundos. Una ejecución satisfactoria desde un árbol limpio deja una marca para ese árbol, y el script de despliegue rechaza un árbol sin ella. Su única excepción es una variable de entorno que imprime una advertencia muy visible, de modo que quien se salta el control en una emergencia tiene que declararlo.

Cómo se aplica en la práctica

La página de créditos del sitio enumera las herramientas de ese control: vnu para HTML, Stylelint, axe-core sobre una página por plantilla, Lighthouse, lychee para enlaces, Prettier, markdownlint, ESLint, Ruff, basedpyright, yamllint, ShellCheck, codespell y gitleaks. Task ejecuta todo bajo un único nombre por tarea, y pre-commit vuelve a ejecutar en cada commit la mitad del conjunto dedicada a comprobar archivos individuales. La página también indica qué queda fuera del control: dos comprobaciones del navegador dirigidas por Puppeteer, el desbordamiento de la ventana gráfica y la regresión visual, que una persona ejecuta bajo demanda.

Un proyecto de una sola persona carece por completo de revisor, que es el punto de partida de una página del portafolio. Las normas tenían que ser ejecutables para que incumplir una hiciera fallar el commit en lugar de esperar a que alguien lo advirtiera. La respuesta, en la etapa de implementación documentada correspondiente a agosto de 2026, fue un control de commit de 22 linters sin nivel de advertencias, además de comprobaciones específicas del proyecto sobre las que ninguna herramienta estándar tiene criterio, como que el árbol desplegado no contenga ninguna ruta ni nombre de host privados y que los textos respetaran las reglas de tono en tres idiomas. Un fallo ya existente se

corrige en la misma pasada en vez de aceptarse. La página expone el coste sin rodeos: confirmar cambios es lento y el control obligatorio es el único revisor del proyecto.

Un control obligatorio también puede contradecirse. Otra página describe un proyecto con un hook de commit y un control completo, un documento que afirmaba que el hook ejecutaba ese control y dos listas de comprobaciones mantenidas a mano en dos archivos. Una prueba que resuelve ambas listas hasta las comprobaciones que realmente invocan mostró cinco comprobaciones que solo se ejecutaban en el control completo, y eran precisamente las que leían los textos. Todas las comprobaciones que protegían el código se ejecutaban automáticamente; todas las que protegían la redacción solo se ejecutaban cuando alguien se acordaba. Las cinco pasaron al hook, dos se aceleraron lo suficiente para ajustarse a su presupuesto de tiempo y la prueba comparativa se mantuvo para detectar diferencias entre las listas.

Dónde surgen los problemas

La mayoría de los controles obligatorios fallan dejando pasar cosas mientras siguen mostrando un resultado satisfactorio.

Las advertencias se acumulan aunque el control dé el visto bueno. Un control que solo falla con errores permite que se acumulen advertencias hasta que el registro es demasiado largo para leerlo, y entonces un problema real llega como la advertencia número doscientos uno. Elimine el nivel de advertencias: o una regla merece hacer fallar el control o se desactiva.

Una comprobación omitida cuenta como aprobada. En GitHub, un estado omitido satisface una comprobación obligatoria, y un trabajo omitido por su condición `if:` comunica un resultado satisfactorio. Un trabajo que depende de otro fallido también se omite y puede no bloquear la fusión. GitHub aconseja usar `always()` con `needs` : exigir un trabajo que siempre se ejecute y que solo apruebe cuando todo aquello de lo que depende haya terminado correctamente.

```
# .github/workflows/check.yml (excerpt)
jobs:
    gate:
        if: always()
        needs: [lint, test, build]
        runs-on: ubuntu-latest
        steps:
```

```
            - run: |
                  for result in ${{ join(needs.*.result, ' ') }}; do
                      test "$result" = success || exit 1
                  done
```

También existe la trampa opuesta: un flujo de trabajo obligatorio omitido por un filtro de rutas o ramas deja sus comprobaciones pendientes y bloquea todas las fusiones. Evite exigir flujos de trabajo que puedan omitirse.

Dos listas divergen. El hook ejecuta un conjunto y el servidor otro, mientras un documento afirma que coinciden. Defina las comprobaciones una sola vez e invoque esa definición desde todos los lugares, o mantenga una prueba que compare las listas y falle cuando difieran.

El control se vuelve lento y la gente lo elude. Siempre hay a mano una opción para saltarse los hooks y, en GitHub, la protección de ramas no se aplica de forma predeterminada a los administradores del repositorio; «Do not allow bypassing the above settings» cambia ese comportamiento al impedir eludir los ajustes anteriores. Mida dónde se consume el tiempo, ejecute en paralelo las comprobaciones independientes, mantenga pequeña la parte que se ejecuta en cada commit y traslade una comprobación realmente lenta a un paso manual documentado.

Una comprobación aprueba porque no comprobó nada. Un linter dirigido a un patrón glob que no coincide con ningún archivo, o un patrón de exclusión que se traga un directorio entero, termina sin errores, y una salida limpia parece lo mismo que un código limpio. Pruebe el propio control: proporcione a cada comprobación un archivo que se sepa que está mal y verifique que la comprobación falla.

Lecturas adicionales

- Introducción a los controles de calidad obligatorios — Condiciones, el control predeterminado Sonar way y la atención al código nuevo.
- Acerca de las ramas protegidas — Comprobaciones de estado obligatorias y quién puede eludirlas.
- Resolución de problemas con las comprobaciones de estado obligatorias — Qué estados cuentan como aprobados y cómo se comportan los flujos de trabajo y trabajos omitidos.

- Fusión automática — La comprobación «Pipelines must succeed» de GitLab y cómo trata los flujos omitidos.
- pre-commit — Comprobaciones declaradas una sola vez e instaladas como hooks de Git para las etapas de commit y push.
- Dependencias de Task — Cómo ejecuta Task las dependencias de una tarea en paralelo y sus comandos en orden.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un control de calidad obligatorio?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/quality-gate/